Trust Relationships in a Mobile Agent System

Hock Kim Tan and Luc Moreau

hkvt99r@ecs.soton.ac.uk, L.Moreau@ecs.soton.ac.uk
Department of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ, UK

Abstract. The notion of trust is presented as an important component in a security infrastructure for mobile agents. A trust model that can be used in tackling the aspect of protecting mobile agents from hostile platforms is proposed. We define several trust relationships in our model, and present a trust derivation algorithm that can be used to infer new relationships from existing ones. An example of how such a model can be utilized in a practical system is provided.

1 Introduction

Mobile agent technology has been identified as a new paradigm that allows flexible structuring of distributed computation over wide-scale networks such as the Internet [11]. One of the main concerns currently impeding the wider acceptance and use of mobile agents, particularly in application areas such as e-commerce [6], is the issue of security. Farmer et. al [4] provides an early discussion of the security problems and requirements unique to mobile agents, as well as the types of security goals that are achievable. A more recent overview of mobile agent security issues, along with a comparative discussion of the current techniques available to address them, can be found in [2], [13] and [8]. In general, we can divide mobile agent security into two broad areas : *host security* (protecting the host platform from a malicious agent) and *code security* (protecting the mobile agent from a malicious host platform).

In this paper, we discuss some of the techniques available for addressing the code security issue and suggest that the manner in which current techniques are implemented may not scale well for a security infrastructure that encompasses a large number of highly mobile agents. We identify trust as an important component of a security infrastructure, and develop a security framework for a mobile agent system which incorporates a simple trust model. Our model is motivated by the similiarities between the manner in which distributed authentication is handled in a public key infrastructure, and the way code security could be handled in a security infrastructure for mobile agents. Existing work on trust relationships within the context of a public key infrastructure is used as a background to define trust relationships specific to a mobile agent system. We then show how new trust relationships may be derived from existing ones in our model and present an algorithm to formalize our approach.

The main contributions of this paper are :

– Identifying trust as an important component in a security infrastructure to handle code security;
– Proposing a security framework which incorporates the notion of trust through the delegation of a code security technique;
– Adapting an existing trust model for a public key certificate system to use in conjunction with the proposed security framework.

An overview of the paper is as follows. In Sect. 2, we discuss current code security techniques and suggest the need to incorporate trust. We develop our trust model and security framework through analogies of the use of trust within a public key infrastructure in Sect. 3. Trust relationships within this model is detailed in Sect. 4, while Sect. 5 describes the trust derivation algorithm that we use. Sect. 6 provides an intuitive discussion of how such a model can be deployed in a mobile agent system. Finally, Sect. 7 concludes the paper with a summary and identifies avenues for possible future work.

2 Mobile Code Security Techniques

Host security is a well researched area for which a number of viable techniques have already been developed. These include mechanisms such as sandbox security in the Java programming language [5], software fault isolation [19], proof carrying code [12] and type safe languages [18]. Code security is however more problematic, since this aspect has only come into prominence recently as a security problem unique to mobile code. Most solutions proposed so far have been conceptual, and it is likely that this area will be crucial in determining the future viability of mobile agent applications in scenarios such as distributed e-commerce.

Some of the more well known code security techniques include code obfuscation [7], encrypted functions [14], tamper-proof hardware [20] and execution tracing [17]. The reader is referred to [8],[9] for a more thorough overview and classification of the code security techniques currently available. Execution tracing, the technique that we employ in the construction of our security framework, involves the detection of unauthorized modifications of an agent through the faithful recording of the agent's behavior during its execution on each host platform. This technique requires each host platform involved to create and retain a log or trace of the operations performed by the agent while resident there. Upon return of the mobile agent, the agent owner may (if she suspects that the mobile agent was not correctly executed) request that the various host platforms submit their individual traces. These are then contrasted against a simulated execution of the original mobile agent (using the information contained in the traces) to detect possible deviations in execution of the agent.

All of these techniques attempt to safeguard the mobile agent with regards to one or more security aspects. For purposes of discussion, we provide a simple classification for the security aspects that these techniques might seek to protect

- *Execution integrity.* This refers to the correct transformation of the current state of the mobile agent to a new state, in accordance with the semantics of the mobile agent code. To accomplish this, it is also necessary to ensure that the correct portion of the code is executed in order to affect the required transformation.
- *State/code integrity.* The state and code of the mobile agent need to be protected from invalid manipulation.
- *State/code visibility.* It may be necessary to permit only certain parts of the state and code to be made visible to the host platform since other parts may contain sensitive information.
- *Resource provision.* It is also important to gurantee the provision of necessary system resources (within the constraints of the resource and security policy of the host platform) to the mobile agent in order for it to execute successfully.

In general, code security techniques only address one or a few security aspects; those that attempt to address every single aspect conceivable are likely to be found deficient in certain aspects upon closer scrutiny. In view of this, it is likely that a future security infrastructure that addresses the code security issue comprehensively will need to incorporate a combination of techniques, rather than a ubiquitous "one-shoe-fits-all" solution. What is therefore required is a mechanism for selecting the appropriate technique or combination of techniques to use, depending on the execution environment and targeted application.

For example, in his discussion of the tamper-proof hardware approach, Yee [22] suggests the use of trust to negate the requirement for hardware to be installed on all execution environments. It could be permissible to run a mobile agent in a software-only environment (using other purely software based code security techniques), if the deployers of the agent have a certain amount of trust in the that environment. Tamper-proof hardware would only need to be installed in environments whose behaviour or reputation is unknown to the deployers.

Certain code security techniques, such as execution tracing, require active intercession on behalf of the agent deployer (the platform deploying the mobile agent will need to verify the execution trace submitted back by the host platform executing the agent). If the deploying platform is the only entity capable of performing such a verification, it will quickly become overwhelmed when the number of mobile agents and corresponding verifications required increase. In order for such a system to scale, the deploying platform must be able to delegate some of its verification activities to other entities in the system. Again, some notion of trust between entities is required for the deploying platform to delegate its activities in this manner.

A more subtle point to be considered is the action of censuring a host platform that has been detected in the act of illegally manipulating some portion of the mobile agent's code or state (i.e. violating state integrity). In most literature describing code security techniques that detect such violations, the assumption is that uniform punitive action is taken towards all perpetrators. On reflection, we see that this inflexibility might not be desirable in every situation. For example,

in e-commerce scenarios, it is possible to acquire additional economic interests or benefits that result from constant interaction with a trusted platform, which may not be readily available from untrusted platforms. Thus when we discover that a trusted platform has violated an agent's integrity, we do not immediately bar our agents from visiting that platform (as might have been the case with an untrusted platform). Instead we could permit migration, but with possibly a more comprehensive code security technique applied. Trust thus provides us with the basis for deciding on a suitable course of action to be taken in dealing with the violation of the agent in the event that such flexibility is advantageous in a given situation.

A large number of agent frameworks, particularly those in e-commerce scenarios, involve the development and evolution of complex trust relationships between the various participating entities. The incorporation of the notion of trust in a security framework allows the development of trust models and metrics to express the nature and flow of trust resulting from the interactions of these entities. Such models and metrics also permit quantitative comparison between different frameworks that may provide useful guidelines on their future development.

We can now identify several points which we believe make a sound argument for the inclusion of the notion of trust as part of an overall security framework when addressing mobile agent code security

- it provides a basis for deciding on the particular code security technique or combination of techniques to be deployed in a particular environment or application;
- it permits the scalability of a system employing certain code security techniques through the delegation of specific security activities;
- it allows flexibility in deciding on the appropriate punitive action to undertake towards perpetrators;
- it allows development of trust models and metrics that express the trust dynamics in e-commerce agent frameworks

The benefits of using incorporating trust and using trust models in a distributed system in general [16] and a mobile agent system in particular [15] have been identified. Certain code security techniques, such as tamper-proof hardware, also incorporate the notion of trust, although in an implicit manner. However, to date, we are not aware of any work that develops a trust model explicitly in the context of a mobile agent system by defining the trust relationships possible in such a system. In the next section, we demonstrate how a simple security framework for a mobile agent system can be developed by adapting the trust model used in a distributed authentication system such as a public key infrastructure.

3 Framework for Code Security in a Mobile Agent System

A public key infrastructure (PKI) [1] is essentially a system that provides all the necessary maintenance activities associated with the complete life cycle of certificates, which are one of the key elements of a distributed authentication service [10]. The main issue in such a service is ensuring that a public key is correctly associated with the identity of an entity that owns the corresponding private portion. PKI systems involve a trusted third party termed a *certificate authority* (CA) that is responsible for verifying name-key bindings through the issuance of certificates. On a large scale basis, a single CA would be incapable of handling the name-key binding activity for all users; thus the need for several CAs arises. In such a situation, an end-user may not be able to immediately identify a certificate received, and may require that the certificate be verified in turn by a CA that he or she is familiar with. This in effect creates a *certification path* through which CAs verify the certificates of other CAs all the way up to a *root authority*, for which a user would be acquainted with. The certification path thus reflects the propagation of trust between different CAs and users in the system.

We can now begin to develop a security framework for a mobile agent system based on the trust model just described. We do not claim that this model is a definitive one as far as mobile agent systems are concerned; rather it provides a guideline on how a more comprehensive model can be developed. Code security techniques have been classified in literature surveys into detection (execution tracing, state appraisal) and prevention mechanisms (code obfuscation, encrypted functions). Prevention mechanisms seek to prevent meaningful manipulation of agent code and hence are the most reliable, although they are usually very complicated and expensive in deployment. They assume a very simplistic trust model; no entity is trusted at all and maximal measures are undertaken to prevent any possible security breach. Detection mechanisms, on the other hand, are more easily deployable since they merely seek to detect possible violations in the agent. More importantly, when such violations are detected, the nature and severity of the violations allow us to determine the different levels of trustworthiness in the platforms concerned. Based on this consideration, we can select an appropriate combination of code security techniques (in addition to the detection mechanism) that needs to be applied on that particular host platform, in line with the original motivations for the use of trust in a code security framework.

With regards to this, we choose to employ execution tracing as our core code security technique in the framework that we are about to describe. This technique is well developed and to date its only criticisms are related to performance and scalability concerns. There are other detection mechanisms available such as forward integrity [22] and state appraisal [3]; execution tracing however offers the important advantage of being able to detect tampering of any part of the agent as opposed to only specific portions, as is the case with the former two mechanisms. To improve scalability, execution tracing requires the introduction of additional entities to undertake the verification process of execution traces

on behalf of the deploying platform (as mentioned in the previous section). In such an instance, these entities assume the role of a trusted third party, not unsimiliar to the role of the CA in a PKI. We refer to this trusted third party as a *verification server*.

In a PKI, the task of associating the public key correctly with an identity (a security requirement for any entity before it can commence utilizing the key), has essentially been delegated from the entity to the CA. In our system, the verification server functions as an intermediary between an agent owner platform (the platform from which the mobile agent is initially launched from) and a host platform. The task of verifying the correct execution of the mobile agent on the host platform has now been delegated from the agent owner platform to the verification server. In addition, verification servers may delegate execution verification activities to other verification servers in the system, analogous to the manner in which CAs verify certificates of other CAs in a PKI.

Trust is generally established with regards to a specific activity, rather than as an unconstrained notion. For example, it is too general to simply state that an entity A trusts another entity B; it would be more accurate to say that A trusts B with respect to a certain activity. The context of trust used in a PKI is generally with respect to the name-key binding activity (other activities may include secure key pair generation, in the event the CA is responsible for key generation as well). In our system, we employ the classification of security aspects defined in Sect. 2 (execution integrity, state/code integrity, etc) as a context for establishing trust. Since we only utilize the execution trace technique, the two main activities would be execution integrity and state/code integrity.

4 A Trust Model for Mobile Agents

One of the seminal papers to discuss the idea of trust relationships in a distributed authentication system is [21]. There are two types of trust relationships introduced in this paper :- direct trust and recommended trust. Direct trust is analogous to the trust obtained between a CA and an entity which generates a public key pair. In this instance, the CA can directly verify the identity of this entity and issue a certificate binding this identity to the entity's public key. Recommended trust is analogous to the trust obtained between an entity that has just received a certificate and the CA that issued it. In this instance, the entity has no way of determining directly that the public key in the certificate is bound correctly to the identity contained within, and trusts the CA to perform this activity for it. A trust derivation algorithm (also presented in [21]) can be used to generate new derived trust relationships from an existing set of direct trust and recommended trust relationships existing in the system.

To achieve a fine grained trust model, we introduce the idea of partitioning a complete mobile agent into several smaller, self-contained state and code components. We believe that in the future, mobile agents will be complex pieces of code composed by their deployers from reusable components that are distributed by third party code producers. In this instance, we can selectively apply differ-

ent code security techniques to different state and code components, and thus establish trust relationships of different contexts with respect to different code and state components. Of course in our model, we only employ the execution trace technique, which we can still apply selectively to different code and state components.

4.1 Trust and belief relationships in a mobile agent system

Before defining the trust and belief relationships in our system, we first define a state space encompassing all the relevant entities

$$\begin{aligned}
\mathcal{OP} &= \{\ A,\ B,\ \ldots\ \} && \text{(Set of agent owner platforms)}\\
\mathcal{VSP} &= \{\ VS_0,\ VS_1,\ \ldots\ \} && \text{(Set of verification servers)}\\
\mathcal{HP} &= \{\ H_0,\ H_1,\ \ldots\ \} && \text{(Set of host platforms)}\\
\mathcal{SC} &= \{\ s_0, s_1,\ \ldots\ \} && \text{(Set of code/state components)}\\
\mathcal{SO} &= \{\ x_0, x_1,\ \ldots\ \} && \text{(Set of security objectives)}
\end{aligned}$$

We assume that all mobile agents in the system can be composed from a combination of predefined set of code and state components made available by third party code producers. Agent owner platforms are platforms where mobile agents are launched from. These agents will migrate through an itinerary of host platforms before terminating or returning to their respective agent owner platforms. Execution tracing of these mobile agents is performed by a set of verification servers distributed throughout the system. Trust and belief relationships between entities in the system are established with respect to a certain type or class of activities; this corresponds to the classification of security aspects mentioned earlier.

(1) VS_0 *trusts.exe* H_0 *with* $(\mathcal{X}, \mathcal{S})$ (Server-host trust)
(2) A *trusts.exe* H_0 *with* $(\mathcal{X}, \mathcal{S})$ (Owner-host trust)
(3) A *trusts.ver* VS_0 *with* $(\mathcal{X}, \mathcal{S})$ (Owner-server trust)
(4) VS_0 *trusts.ver* VS_1 *with* $(\mathcal{X}, \mathcal{S})$ (Server-server trust)
(5) A *believes.exe* H_0 *with* $(\mathcal{X}, \mathcal{S})$ (Owner-host belief)
(6) A *believes.ver* VS_0 *with* $(\mathcal{X}, \mathcal{S})$ (Owner-server belief)

where $A \in \mathcal{OP},\ \mathcal{X} \in \mathcal{SO},\ \mathcal{S} \subseteq \mathcal{SC},\ VS_0, VS_1 \in \mathcal{VSP},\ H_0 \in \mathcal{HP}$

Fig. 1. Basic trust and belief relationships

We now describe the basic trust and belief relationships (Figure 1) that are possible between the three different types of entities in the system (host platforms, agent owner platforms and verification servers). The first relationship, server-host trust, represents the trust that a verification server, VS_0, has in a host platform, H_0, to undertake the transformation of the state components specified in $\mathcal{S}$ correctly with respect to a security objective $\mathcal{X}$. $\mathcal{S}$ and $\mathcal{X}$ are thus constraints on the context for which this relationship is applicable. We give this type of trust

relationship the term *execution trust*. A server-host trust relationship is initially established when a verification server successfully validates the execution trace submitted by a host platform. This execution trace pertains to a mobile agent composed of the components $\mathcal{S}$, undertaken by the verification server to test the reliability of the host platform in question.

Owner-host relationship represents the trust an agent owner platform, A, has in a host platform, H_0, to undertake the transformation of the code/state components specified in $\mathcal{S}$ correctly with respect to a security objective $\mathcal{X}$. Since verification servers are the only entities in our system possessing the functionality necessary to ascertain execution correctness, this type of relationship cannot be established directly by an agent owner platform. It can only be derived from other existing trust relationships, as will be demonstrated later.

Owner-server trust relationship refers to when an agent owner platform, A, trusts a verification server, VS_0 to undertake correct verification of the transformation of state components specified in $\mathcal{S}$, with respect to a security objective $\mathcal{X}$. This trust relationship is analogous to the idea of recommendation or recommended trust as described in [21]. In our context, we shall give it the term *verification trust*. Verification trust is established directly when an agent owner platform decides to delegate the task of verifying correct execution of its mobile agents to a verification server. Once such a relationship is in place, the agent owner platform will supply the verification server with the necessary information and resources (for example, a copy of all mobile agents launched by the agent owner) to undertake the verification successfully.

Server-server trust relationship is interpreted to mean that a verification server, VS_0, trusts another verification server, VS_1 to undertake verification of the correct transformation of state components, specified in $\mathcal{S}$, with respect to a security objective $\mathcal{X}$. This could result from a verification server delegating the responsibility of verifying certain host platforms to other verification servers in the system, and is thus a relationship that can be established directly.

All of these trust relationships, with the exception of owner-host trust, express trust on the basis of explicit actions (or the results of those actions) undertaken by entities involved in the relationship. Sect. 6 elaborates further on how these relationships are initially established and how they might evolve over a period of time. We also require another type of relationship to express the assumptions and/or beliefs that an entity has about another entity in the system. For example, an agent owner platform, A, may have reason to believe (based on knowledge acquired from an external source) that host platform $\mathcal{H}_0$, is capable of executing $\mathcal{S}$ correctly with respect to $\mathcal{X}$. A is not capable of directly verifying the accuracy of this belief (since only verification servers possess the functionality necessary to verify execution traces from host platforms). This belief is thus expressed in the form of a owner-host *belief relationship* (statement 5). The idea of a server-host and a server-server belief relationship is equally valid in this context; they are not included in order to simplify the trust derivation algorithm that we develop in the next section.

It is important to note that trust and belief relationships are not symmetric in our system (i.e VS_0 *trusts.ver* VS_1 does not necessarily imply that VS_1 *trusts.ver* VS_0). Also, we did not introduce the idea of trust originating from a host platform (i.e. the host platform is the terminating point for an execution trust relationship). This could be useful if we wish to extend our model to encompass the issue of host security (i.e. the host needs to be able to trust that agent owner platforms do not dispatch malicious agents to its environment), but we do not address this here.

In the next section, we explore how we can combine these basic trust and belief relationships to produce new trust and belief relationships and present a trust derivation algorithm to demonstrate our approach.

5 Deriving New Trust Relationships

The different ways in which new trust and belief relationships can be formed from existing ones is illustrated in Figure 2. For the server-host and server-server trust derivations, the intersection of the code/state component constraint sets $\mathcal{S}_1$ and $\mathcal{S}_2$ in the derived relationship, indicates that the derived relationship should be in the context of the code/state components common to both relationships that it is derived from. Owner-host and owner-server trust derivations will, in addition, involve owner-host and owner-server belief relationships as well. In this case, we only derive a new trust relationship if there already exists a belief relationship between the agent owner platform and the host platform or verification server concerned. Again, this new relationship will be in context of the code/state components common to both the two initial relationships as well as the belief relationship ($\mathcal{S}_1 \cap \mathcal{S}_2 \cap \mathcal{S}_3$).

Note that we do not include the idea of inferring new trust or belief relationships by using an existing belief relationship as a starting point. This is due to the fact that while trusting behaviour is transitive (resulting for example, from delegation of the verification activity in the case of server-server or server-host trust derivations), trusting belief is not. However the context of an existing belief relationship can be altered independently by an agent owner platform depending on the results of the new trust relationships derived. For example, if in deriving a new owner-host trust relationship, $\mathcal{S}_1$ and $\mathcal{S}_2$ are both supersets of $\mathcal{S}_3$, then the agent owner platform could choose to expand the constraints of its current belief relationship with the host platform from $\mathcal{S}_3$ to $\mathcal{S}_1 \cup \mathcal{S}_2$ instead. This ensures that the next time a trust relationship is derived from the existing owner-server and server-host relationships, a wider constraint can be achieved.

5.1 Verification path

It is important to note the exact sequence in which the existing relationships are combined to provide a context for the new trust relationships obtained. To achieve this, the idea of a verification path is introduced. A verification path refers to a sequence of entities (verification servers, agent owner platforms or

Deriving server-host trust

if there exists relationships of the form
a) VS_1 *trusts.exe* H_0 *with* $(\mathcal{X}, \mathcal{S}_1)$ b) VS_0 *trusts.ver* VS_1 *with* $(\mathcal{X}, \mathcal{S}_2)$
then we can infer a new relationship of the form
VS_0 *trusts.exe* H_0 *with* $(\mathcal{X}, \mathcal{S}_1 \cap \mathcal{S}_2)$

Deriving server-server trust

if there exists relationships of the form
a) VS_0 *trusts.ver* VS_1 *with* $(\mathcal{X}, \mathcal{S}_1)$ b) VS_1 *trusts.ver* VS_2 *with* $(\mathcal{X}, \mathcal{S}_2)$
then we can infer a new relationship of the form
VS_0 *trusts.ver* VS_2 *with* $(\mathcal{X}, \mathcal{S}_1 \cap \mathcal{S}_2)$

Deriving owner-host trust

if there exists relationships of the form
a) VS_0 *trust.exe* H_0 *with* $(\mathcal{X}, \mathcal{S}_1)$ b) A *trusts.ver* VS_0 *with* $(\mathcal{X}, \mathcal{S}_2)$
and if there exists a belief relationship of the form
c) A *believes.exe* H_0 *with* $(\mathcal{X}, \mathcal{S}_3)$
then we can infer a new trust relationship of the form
A *trusts.exe* H_0 *with* $(\mathcal{X}, \mathcal{S}_1 \cap \mathcal{S}_2 \cap \mathcal{S}_3)$

Deriving owner-server trust

if there exists relationships of the form
a) A *trusts.ver* VS_0 *with* $(\mathcal{X}, \mathcal{S}_1)$ b) VS_0 *trusts.ver* VS_1 *with* $(\mathcal{X}, \mathcal{S}_2)$
and if there exists a belief relationship of the form
c) A *believes.ver* VS_1 *with* $(\mathcal{X}, \mathcal{S}_3)$
then we can infer a new trust relationship of the form
A *trusts.ver* VS_1 *with* $(\mathcal{X}, \mathcal{S}_1 \cap \mathcal{S}_2 \cap \mathcal{S}_3)$

Fig. 2. Deriving new trust relationships

host platforms) involved in the derivation of new trust relationships. Consider for example, the following trust relationship statements

(a) VS_0 *trusts.ver* VS_1 *with*...
(b) VS_1 *trusts.ver* VS_2 *with* ...
(c) VS_2 *trusts.exe* H_0 *with* ...
(a) and (b) can be combined to obtain a new trust relationship :
(d) VS_0 *trusts.ver* VS_2 *with* ...

The verification path at this stage involves the entities VS_0, VS_1 and VS_2 in that given sequence. d) and c) can subsequently be combined to obtain a new trust relationship :

(e) VS_0 *trusts.exe* H_0 *with* ...

The verification path sequence now involves the entities VS_0, VS_1, VS_2 and H_0. Further derivation of new trust relationships from (e) will involve expanding

the verification path in a similiar manner. The idea of a verification path will be used in the algorithm that we develop next.

5.2 Trust Derivation Algorithm

We can now detail a trust derivation algorithm (Figure 3), which we extend from the one presented in [21] by incorporating the idea of using beliefs in the process of deriving new trust relationships. The algorithm demonstrates how the derivations just explained can be systematically applied in a system described by an initial set of trust relationships. The goal of this algorithm is to generate from this initial set of trust relationship expressions, a set of tuples $\mathcal{HS}$, that describe owner-host trust relationships that exist between any given agent owner platform, A, in the system and all other host platforms in the system. The algorithm works on the elements within two sets, $\mathcal{HS}$ and $\mathcal{N}$.

$\mathcal{HS}$ is a set of tuples, with each tuple consisting of a host platform, H, as well as a set of state components, C. Initially, $\mathcal{HS}$ is empty and the algorithm will append elements to it during its execution. At the termination of the algorithm, each tuple in $\mathcal{HS}$ represents a new host platform, H, in which the agent owner platform A can establish a new derived owner-host trust relationship with the form A *trusts.exe* H *with* $(\mathcal{X}, C)$.

$\mathcal{N}$ is also a set of tuples, each tuple representing a possible next step in a verification path. The constituent components of each tuple are:

- a verification server, VS_i, which is the next possible entity in a verification trust path;
- a sequence $seq = [A, VS_1, VS_2 ..., VS_i]$ which represents the sequence of the verification path traversed so far;
- a set of code/state components $\mathcal{S}c$ which represents the code/state components for which trust is still applicable on the given verification path.

The expression $seq \bullet VS_j$ is used to indicate that a new entity VS_j is being appended to a sequence seq of a verification path. At the start, $\mathcal{N}$ is initialized with the tuples that correspond to all initial trust relationships of the form A *trusts.ver* VS_i *with* $(\mathcal{X}, \mathcal{S}c_i)$, where A is the current agent owner platform to which the algorithm is being applied to.

5.3 Trust Derivation Algorithm - Example

Consider a system consisting of a set of host platforms, **HE**, verification servers, **VE**, code/state components **SE** and a single agent owner platform, A.

$\mathbf{VE} = \{VS_a, VS_b, VS_c, VS_d\}$
$\mathbf{HE} = \{H_p, H_q\}$
$\mathbf{SE} = \{s_0, s_1, s_2, s_3, s_4\}$

Signature
$C = \mathbb{P}(\mathcal{SC})$
$HostTuple = H \times C$
$\mathcal{HS} = \mathbb{P}(HostTuple)$
$\mathcal{HS} ::= \{\langle H_1,\ C_1\rangle,\ \langle H_2,\ C_2\rangle, \ldots\}$
$seq = Ownerplatform \times VS \times \ldots \times VS$
$PathTuple = VS \times seq \times \mathcal{S}c$
$\mathcal{N} = \mathbb{P}(PathTuple)$
$\mathcal{N} ::= \{\langle VS_1,\ seq_1, \mathcal{S}c_1\rangle,\ \langle VS_2,\ seq_2, \mathcal{S}c_2\rangle, \ldots\}$

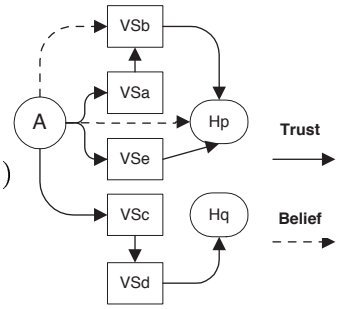
Initialisation
$\mathcal{TBS}$ = { Set of initial trust and belief relationships }
$\mathcal{HS} = \{\}$
$\mathcal{N} = \{\langle VS_1, [A, VS_1], \mathcal{S}c_1\rangle, \langle VS_2, [A, VS_2], \mathcal{S}c_2\rangle, \ldots, \langle VS_j, [A, VS_j], \mathcal{S}c_j\rangle\}$
where A is the current agent owner platform to which the algorithm is being applied and $VS_1, VS_2, \ldots, VS_j$ are all the verification servers for all trust relationships A *trusts.ver* VS_i *with* $(\mathcal{X}, \mathcal{S}c_i) \in \mathcal{TBS}$

boolean *foundtuple*;
Do until $\mathcal{N} = \emptyset$:
Select a *PathTuple* $\langle VS_i, seq_i, \mathcal{S}c_i\rangle$ from $\mathcal{N}$
for every VS_i *trusts.exe* H_k *with* $(\mathcal{X}, \mathcal{S}_l) \in \mathcal{TBS}$
if A *believes.exe* H_k *with* $(\mathcal{X}, \mathcal{S}_m) \in \mathcal{TBS}$
begin
foundtuple = *false*
for every *HostTuple* $\langle H_j, C_j\rangle$ in $\mathcal{HS}$
if $H_k = H_j$
$C_j := C_j \cup (\mathcal{S}_l \cap \mathcal{S}_m \cap \mathcal{S}c_i)$, *foundtuple* = *true*
if *not foundtuple*
$\mathcal{HS} := \mathcal{HS} \cup \langle H_k, \{\mathcal{S}_l \cap \mathcal{S}_m \cap \mathcal{S}c_i\}\rangle$
end
end
for every VS_i *trusts.exe* VS_n *with* $(\mathcal{X}, \mathcal{S}_p) \in \mathcal{TBS}$
if A *believes.ver* VS_n *with* $(\mathcal{X}, \mathcal{S}_q) \in \mathcal{TBS}$
if $VS_n \notin seq_i$
$\mathcal{N} := \mathcal{N} \cup \langle VS_n,\ seq_i \bullet VS_n, \{\mathcal{S}_p \cap \mathcal{S}_q \cap \mathcal{S}c_i\}\rangle$
end
$\mathcal{N} := \mathcal{N} \setminus \langle VS_i, seq_i, \mathcal{S}c_i\rangle$

Fig. 3. Trust derivation algorithm

At the start, we assume that the system already has the following initial trust and belief relationships :

1. A *trusts.ver* VS_a *with* $(\mathcal{X}, \mathbf{SE})$
2. A *trusts.ver* VS_c *with* $(\mathcal{X}, \mathbf{SE})$
3. A *trusts.ver* VS_e *with* $(\mathcal{X}, \mathbf{SE})$
4. A *believes.ver* VS_b *with* $(\mathcal{X}, \{s_1, s_2\})$
5. A *believes.exe* H_p *with* $(\mathcal{X}, \{s_2, s_3\})$
6. VS_a *trusts.ver* VS_b *with* $(\mathcal{X}, \{s_0, s_1, s_2\})$
7. VS_b *trusts.exe* H_p *with* $(\mathcal{X}, \{s_1, s_2\})$
8. VS_c *trusts.ver* VS_d *with* $(\mathcal{X}, \{s_1, s_4\})$
9. VS_d *trusts.exe* H_q *with* $(\mathcal{X}, \mathbf{SE})$
10. VS_e *trusts.exe* H_p *with* $(\mathcal{X}, \{s_3, s_4\})$



We now proceed to apply the algorithm to A to determine all the new trust relationships that can be derived with the host platforms in the system. We start by initializing $\mathcal{N}$ with all the trust relationships that originate from the agent owner platform A (1, 2 and 3).

$\mathcal{N} = \{\langle VS_a,\ [A, VS_a],\ \mathbf{SE}\rangle,\ \langle VS_c,\ [A, VS_c],\ \mathbf{SE}\rangle,\ \langle VS_e,\ [A, VS_e],\ \mathbf{SE}\rangle\}$

$\mathcal{HS} = \{\}$

After the first pass of the algorithm, we have

$\mathcal{N} = \{\langle VS_b,\ [A, VS_a, VS_b],\ \{s_1, s_2\}\rangle,\ \langle VS_c,\ [A, VS_c],\ \mathbf{SE}\rangle,\ \langle VS_e,\ [A, VS_e],\ \mathbf{SE}\rangle\}$

$\mathcal{HS} = \{\}$

After the second pass of the algorithm, we have

$\mathcal{N} = \{\langle VS_c,\ [A, VS_c],\ \mathbf{SE}\rangle,\ \langle VS_e,\ [A, VS_e],\ \mathbf{SE}\rangle\}$

$\mathcal{HS} = \{\langle H_p, \{s_2\}\rangle\}$

After the third pass of the algorithm, we have

$\mathcal{N} = \{\langle VS_e,\ [A, VS_e],\ \mathbf{SE}\rangle\}$

$\mathcal{HS} = \{\langle H_p, \{s_2\}\rangle\}$

After the fourth pass of the algorithm, we have

$\mathcal{N} = \{\}$

$\mathcal{HS} = \{\langle H_p, \{s_2, s_3\}\rangle\}$

Thus we can form a new trust relationship of the form

A *trusts.exe* H_p *with* $(\mathcal{X}, \{s_2, s_3\})$

6 Deploying the Framework

We discuss intuitively how the proposed framework could be deployed in a mobile agent system. Consider a community of host platforms, verification servers and agent owner platforms with trust relationships already established among themselves. A new agent owner platform that wishes to participate in the community will need to establish trust relationships with one or more verification servers. Through a short interaction with a selected verification server, the agent owner platform could determine the host platforms that the server in question has a trust relationship with. The agent owner platform can initially ascertain the reliability of that verification server by composing a mobile agent and launching it to a host platform, with the execution trace being submitted back to both

the agent owner platform and the verification server. The results reported back by the verification server are checked for consistency with the validation of the trace by the agent owner platform.

Once the agent owner platform is satisfied, it establishes a trust relationship with the verification server (with respect to specific components) and executes the trust derivation algorithm to obtain new trust relationships with other host platforms. This provides it with a potential itinerary for future mobile agents that it wishes to launch (in the event that the agent owner platform supplies a predefined itinerary), or useful information that can be embedded in the agent itself (should the agent be capable of dynamically deciding its itinerary while it migrates). New verification servers that join the community can establish trust relationships with existing verification servers in a similar manner. New host platforms on the other hand, could advertise their presence through a registry service after which they can be tested for reliability in hosting mobile agents by verification servers who express interest in establishing trust relationships with them.

The key to the evolution of the trust relationships in this framework is the verification of an execution trace submitted by a host platform to a verification server. Trust relationships are initially established as described above, and remain static as long as all traces checked are valid. The moment a verification server detects an invalid trace, the nature of its current trust relationships with the offending host platform will be altered. This could range over several possible alternatives : severing all existing trust relationships, severing some trust relationships or degrading existing relationship(s) by reducing the number of components that the relationship(s) is valid for.

Returning to the analogy that we introduce at the start of Sect. 3 (i.e. the verification server being roughly equivalent to a CA in a PKI), we note that the CA is trusted to maintain the integrity of the key-to-name binding within a certificate. Certificate revocation is employed when such integrity becomes suspect (for example, due to a suspected key compromise before the expiry of the certificate). This is typically implemented using a periodic publication mechanism such as certificate revocation lists (CRLs), which can be accessed by other entities that need to validate certificates. A verification server, on the other hand, is trusted to maintain the integrity of mobile agent execution, and alters its trust relationship (degradation or destruction) with the offending platform when a violation of this integrity is detected. Information about this relationship change (*trust information*) is then propagated to all other verification servers or agent owner platforms (the trustors) that have established trust relationships with the server that detected the violation (the trustee). This will in turn result in a corresponding change in trust relationships on those servers and platforms as well. In effect, trust information is equivalent to a CRL in a PKI, with the difference that trust information is propagated instead of being published. Agent owner platforms could use the event of reception of trust information as a trigger to execute the trust derivation algorithm again in order to recalculate their new trust relationships with existing host platforms.

The actual mapping between the detection of a violation in an execution trace and the subsequent change affected in a trust relationship (destruction or degradation) is a function of a security policy which can be either administered locally or globally administered. The discussion of such policies and their implication on the trust dynamics of the system as a whole is beyond the scope of this paper, but remains important work to be accomplished in studying the effects of a trust model in a mobile agent security framework.

7 Conclusion

This paper proposes the incorporation of a trust model as part of a security framework for mobile agents. We argue that the notion of trust can aid in a more flexible and scalable deployment of existing code security techniques. We also suggest that the manner in which trust would be employed in a wide scale security infrastructure for mobile agents has many parallels to the way it is used currently in a distributed authentication system such as a public key infrastructure. Based on this motivation, we propose a simple security framework for a mobile agent system that resembles the structure of a public key infrastructure. Drawing from existing work on trust relationships in such an infrastructure, we define several trust relationships for a mobile agent system. We then demonstrate how new trust relationships could be derived from existing ones, and present an algorithm to formalize our approach.

We believe that the material developed here is representative of the initial work required in the construction of a complete trust model for a mobile agent system. Such a model would permit a detailed insight into the complex interactions that involve trust in a mobile agent system. Future work in this direction could involve a more precise and formal definition of trust relationships (including, for example, explicit negative trust relationships) specific to mobile agents. There will also be a need to investigate how execution tracing (and other existing code security techniques) could be modified to fit effectively within the structure of such a framework.

References

1. C. Adams and S. Lyold. *Understanding Public Key Infrastructure : Concepts, Standards and Deployment Considerations.* Macmillan Technical Publishing, 1999.
2. D. M. Chess. Security Issues in Mobile Code Systems. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.
3. W. Farmer, J. Guttman, and V. Swarup. Security for mobile agents : Authentication and state appraisal. In *European Symposium on Research in Computer Security*, number 1146 in LNCS. Springer-Verlag, 1996.
4. W. Farmer, J. Guttman, and V. Swarup. Security for mobile agents: Issues and requirements. In *Computer Communications, Special Issue on Advances in Research and Application of Network Security*, October 1996.
5. L. Gong. Java Security Architecture (JDK1.2). Technical report, Sun Microsystems, March 1998.

6. R. H. Guttman, A. G. Moukas, and P. Maes. Agents as mediators in electronic commerce. *Electronic Markets*, 8(1), May 1998.
7. F. Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.
8. W. Jansen. Countermeasures for Mobile Agent Security. In *Computer Communications, Special Issue on Advances in Research and Application of Network Security*, November 2000.
9. N. Karnik. *Security in Mobile Agent Systems*. PhD thesis, Department of Computer Science and Engineering, University of Minnesota, 1998.
10. B. Lampson, M. Abadi, M. Burrows, and E.Wobber. Authentication in distributed systems : Theory and practice. *ACM Transactions on Computer Systems*, 10(4), November 1992.
11. D. B. Lange and M. Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3), 1999.
12. G. Necula and P. Lee. Safe kernel extensions without run-time checking. In *Proceedings of the 2nd Symposium on Operating System Design and Implementation (OSDI '96), Washington*, October 1996.
13. R. Oppliger. Security issues related to mobile code and agent-based systems. *Computer Communications*, 22(12), July 1999.
14. T. Sander and C. F. Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.
15. K. Schelderup and J. Ølnes. Mobile agent security - issues and directions. In *Proceedings of the 6th International Conference on Intelligence and Services in Networks, Barcelona, Spain*, April 1999.
16. V. Swarup and J .T. Fabrega. Trust : Benefits, models and mechanisms. In *Secure Internet Programming : Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS. Springer-Verlag, 1999.
17. G. Vigna. Cryptographic traces for mobile agents. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.
18. D. Volpano and G. Smith. Language Issues in Mobile Program Security. In *Mobile Agents and Security*, number 1419 in LNCS. Springer-Verlag, 1998.
19. R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Efficient software-based fault isolation. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles, pages 203–216*, December 1996.
20. U. G. Wilhelm, S. Staamann, and L. Buttyan. Introducing trusted third parties to the mobile agent paradigm. In *Secure Internet Programming : Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS. Springer-Verlag, 1999.
21. R. Yahalom, B. Klein, and T. Beth. Trust relationships in secure systems—A distributed authentication perspective. In *Proceedings of the IEEE Symposium on Research in Security and Privacy, pages 150–164*, 1993.
22. B. S. Yee. A sanctuary for mobile agents. In *Secure Internet Programming : Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS. Springer-Verlag, 1999.